August 1973

A PRELIMINARY QLISP MANUAL

by

René Reboh

and

Earl Sacerdoti

Artificial Intelligence Center
Technical Note 81
SRI Project 8721

The research reported herein was sponsored primarily by the National Aeronautics and Space Administration under Contract NASW-2086. This work was also supported in part by the Advanced Research Projects Agency through Contract DAHC04-72-C-0008.

ABSTRACT

A preliminary version of QLISP is described. QLISP permits free intermingling of QA4-like constructs with INTERLISP code. The preliminary version contains features similar to those of QA4 except for the backtracking of control environments. It provides several new features as well.

This preliminary manual presumes a familiarity with both INTERLISP and the basic concepts of QA4. It is intended to update rather than replace the existing documentation of QA4.

I INTRODUCTION

QLISP brings together in a natural fashion the control structures, pattern matching, and net storage mechanisms of QA4*[1], and the versatility, programming ease, and interactive features of INTERLISP[2] (formerly called BBN-LISP). The system permits free intermixing of QA4, CLISP[3], and the INTERLISP code so that, for example, users may write programs or type expressions for evaluation in LISP, CLISP, or QA4, or in a mixture of all three.

QLISP has been implemented by means of the error correction facility in INTERLISP. A valid LISP expression will never be seen by the QLISP processor. Thus, programs that do not use QA4 constructs will run as fast in QLISP as in LISP. When the LISP interpreter encounters an ill-formed expression, it calls an error routine that in turn invokes the error analyzer. If the form is recognized as a QA4 construct, it is translated to an equivalent LISP form that is returned to the interpreter for evaluation; if the expression is a valid CLISP construct, a similar translation takes place.

In either case, the translation is stored with the original QLISP expression so that the analysis and translation are done only once.

Since QLISP allows QA4 operations to be embedded directly in LISP, the new system does not require the QA4 interpreter or stack mechanism. Furthermore, QLISP programs are stored as standard LISP functions rather than as expressions in the QA4 discrimination net, thus reducing the search time required for associative retrievals as well as for function calls.

---

* Superscripts denote references listed at the end of the report.

A preliminary version of QLISP is now available. It provides features similar to those of QA4 except for the backtracking of control environments. It provides several new features as well.

The major differences between this version of QLISP and the current QA4 are:

- QLISP functions are defined and stored in the same way as LISP functions. Functions are stored as values of net variables in QA4.
- The QA4 data-type SET has been renamed CLASS, and a new data-type VECTOR has been added.
- The QA4 SETQ and SETQQ statements have been renamed MATCHQ and MATCHQQ. In addition, QLISP provides a MATCH statement (the pattern-matching equivalent of the LISP function SET).
- The QA4 statement EXISTS has been renamed IS, and a corresponding statement ISNT has been added.
- The syntax of the net storage and retrieval statements has been modified.
- "Demons" have been replaced by "teams" of functions that may be applied by any net storage or retrieval function.
- Tuples are now equivalent to LISP lists. The external form of (TUPLE a_1 a_2 ... a_n) is now (a_1 a_2 ... a_n) rather than (TUPLE a_1 a_2 ... a_n). The internal form of (a_1 a_2 ... a_n) is now (TUPLE a_1 a_2 ... a_n) rather than (APPL a_1 (TUPLE a_2 a_3 ... a_n)).
- Processes are not currently available.
- Backtracking out of the scope of a single statement, e.g., backtracking to a previous IS, MATCH, or GOAL statement is not currently possible. However, as a temporary expedient, the effect of backtracking to an IS can be simulated by using BIS ("Backtrack IS").

This is a preliminary QLISP manual. It is intended to update rather than to replace the existing documentation of QA4. <u>It presumes a familiarity with both INTERLISP and QA4</u>.

We are convinced that QLISP can be the most usable of the recent generation of AI languages. With this in mind, the authors hereby elicit and encourage feedback from the user community.

A sample QLISP program is given in Annex A. The syntax of QLISP statements is given in Appendix B.

II THE QLISP LANGUAGE

A. QLISP Expressions

QLISP provides complete freedom in intermingling LISP expressions with those that provide net storage and retrieval, pattern matching, and control structure manipulation. As an example, consider the ARE-COUSINS program:

```
(QLAMBDA (←PERSON1 ←PERSON2)
         (IS (FATHER $PERSON1 ← F))
         (IS (UNCLES $PERSON2 ←← U))
         (IF (MEMB $F $U)
             THEN (PRINT (' ($PERSON1 AND $PERSON2 ARE COUSINS)))
             ELSE (PRINT (' ($PERSON1 AND $PERSON2 ARE NOT COUSINS)))))
```

When this program is executed, two associative retrievals from the discrimination net will obtain the father of the first person and the uncles of the second person. If the father of the first person is among the uncles of the second, we proclaim the two persons to be cousins.

B. The QLISP Discrimination Net

QLISP provides data base storage and retrieval facilities similar to those of QA4. Data of type TUPLE, VECTOR, BAG, or CLASS may be stored in a discrimination net and accessed by associative retrieval.

A tuple is equivalent to a LISP list. The form (f a_1 a_2 ... a_n) will be stored internally as (TUPLE f a_1 a_2 ... a_n) unless f is declared to take a VECTOR, BAG, or CLASS as its argument. Such a declaration is performed by evaluating DEFTYPE[f;type], where f is a function or predicate name and type is VECTOR, BAG, or CLASS.

A vector is similar to a tuple, that is, it contains an ordered

sequence of elements. The treatment of tuples and vectors differs only if they are evaluated. The value of a tuple is the result of applying the function specified by its first element to the values of the rest of its elements; the value of a vector is simply a vector of the values of its elements. Thus a tuple is useful for representing an evaluable form containing a function and its arguments, whereas a vector is useful for representing an argument list alone.

A bag is a collection of unordered elements, and the elements may be duplicated, that is (BAG A A B C) is EQ to (BAG A C B A).

A class is a collection of unordered elements, without duplication, that is, (CLASS A B C A) is EQ to (CLASS C B A).

C. Constructing QLISP Expressions

The QLISP functions TUPLE, VECTOR, BAG, and CLASS cause an expression of the appropriate type to be built and dropped into the discrimination net. For example, (BAG F A B) will create a bag with elements A, B, and F. Similarly, (TUPLE F A B) will create the tuple (F A B) regardless of whether F was DEFTYPEd.

The construction of new expressions is simplified by the use of the STRIP operator(!). The STRIP operator causes a level of parentheses to be stripped from its argument (and thus is meaningless at the top level).

For example, (VECTOR (!(VECTOR A B))(!(VECTOR B C))) will cause the vector (VECTOR A B B C) to be built.

Note that (! $X) is equivalent to $$X.

D. Inverse Quote Mode, Evaluation, and Instantiation

Statements for net storage and retrieval, pattern matching, and control structure manipulation are interpreted in inverse quote mode. That

is, atoms that are not otherwise identified are treated as constants. Variables are indicated by a prefix character or characters.

These statements normally instantiate their arguments rather than evaluate them. To instantiate an expression is simply to replace its net variables by their values. For example, if the variable $X is bound to B, then the QLISP statement (ASSERT (FOO A $X)) will assert the expression (FOO A B), not the result of evaluating (FOO A B).

The "at" sign (@) is used to force evaluation of the following expression. For example, (ASSERT (@ (FOO A $X))) will assert the result of evaluating (FOO A B).

The quote mark(') is used to force instantiation where evaluation would normally take place. For example, (PRINT ('(THIS EXPRESSION WILL $X INSTANTIATED))) will cause (THIS EXPRESSION WILL B INSTANTIATED) to be printed.

The colon prefix (:) is used to prevent the instantiation of a $ variable when it occurs in an expression to be instantiated. For example, (PRINT ('(THE VALUE OF :$X IS $X))) will cause (THE VALUE OF $X IS B) to be printed.

E. QLISP Functions

QLISP functions are of three varieties: LAMBDA and NLAMBDA, as in INTERLISP, and QLAMBDA, which is equivalent to the QA4 LAMBDA and has a pattern as its bound variable part. QLAMBDA expressions admit "implicit PROGNs" just as LAMBDAs and NLAMBDAs do.

F. Declaring Local Variables

The QPROG statement is an analog of the LISP PROG feature. It allows the declaration of net variables in the local context as well as

local LISP variables. The format of the QPROG statement is:

(QPROG args $e_1 \ldots e_n$).

LISP variables are denoted in the args list exactly as in the LISP PROG statement. Net variables are denoted by prefixing them with a left arrow (←).

The expressions e_i are arbitrary QLISP expressions.

For example, the statement

```
(QPROG (U ←V (W Ø) (←X (TUPLE)))
        .
        .
        .
        )
```

will cause local variables U,V,W, and X to be declared. U and W will be LISP variables. V and X will be net variables. U will have an initial value of NIL, V's initial value will be NOSUCHPROPERTY W will be bound to Ø, and X will be bound to the empty tuple.

Both PROG and QPROG should be exited by either of the functions RETURN or QRETURN. QRETURN is similar to the LISP RETURN, except that it instantiates its argument rather than evaluating it.

G. QLISP Backtracking

The side effects of QLISP computations may be undone (in the INTER-LISP sense) by the use of the QLISP failure mechanism. A statement that invokes pattern matching will fail if no match exists or if all matches have been exhausted. Other statements may be caused to fail by the use of the FAIL statement which is described below.

A failure will cause a return to some backtrack point and undo all undoable computations performed since the backtrack point was established.

Manipulation of expressions in the net is undoable unless it is

done with respect to the context ETERNAL, or one of its descendants. Manipulation of list structures is undoable if it is done by means of "/ functions."[4] In addition to the functions provided by INTERLISP, QLISP has a /SETQ function as well.

Backtrack points are established within all net storage and retrieval statements and within QLAMBDA expressions that have the BACKTRACK option.

Failures may be caused explicitly by executing the FAIL statement. Its format is (FAIL name).

If name is absent or NIL, FAIL causes a failure.

If name is CALLER, FAIL causes the last net storage or retrieval statement to fail.

If name matches the NAME of a net storage or retrieval statement, FAIL causes the named statement to fail. (Assignments of NAMEs to statements and examples of the use of the FAIL statement will be discussed in the following section.)

H. Statements of the QLISP Language

A full listing of the syntactic form of each non-LISP statement appears in Appendix B. New statements and those that differ significantly from their old QA4 counterparts are discussed below.

In the syntactic forms to follow, braces ({}) indicate an optional clause. The ordering of options is arbitrary.

The net storage and retrieval statements have a new common syntax. First, the syntax will be discussed in general, and then each of these statements will be described.

1. Syntax of Net Storage and Retrieval Statements

The general form of a QLISP net storage and retrieval statement is:

(statement-type p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$})

The statement-type is one of the following:

QPUT
QGET
ASSERT
DENY
IS
ISNT
BIS
INSTANCES
GOAL
CASES
DELETE

p-exp is a pattern to be instantiated.

The team must instantiate to a class, a bag, or a tuple of QLAMBDA functions. They will be applied to the instantiation of p-exp. If a failure occurs during the application of one function in the team, its side effects are undone and the next function in the team is tried. With the exception of the GOAL statement, the application of the team functions is for their effect rather than their values (i.e., the values returned by the team functions are never used by the calling statement).

ctx, when present, must instantiate to a context. This context will be passed as a context recommendation to the functions in the team (i.e., it will be used as a default value for context references in the team functions). The following are the possible context specifications and their meanings:

- LOCAL--Current context.
- GLOBAL--Top context.
- ETERNAL--Current default context. Changes made in this context will not be backtrackable.
- UNIVERSAL--Top context. Changes made in this context will

not be backtrackable.

- A variable that was previously bound to a context.
- A CONTEXT statement.

If the WRT option is omitted in the statement, the context is bound by default to the last context recommendation from a calling statement. If no such recommendation was made, the top context is used.

The statement may be given a name. A team function may refer to that name in a FAIL statement. The default name is the type of the statement itself (e.g., ASSERT).

ind_1 $prop_1$ ... ind_n $prop_n$ are property specifications. Their use in each statement type is discussed below.

2. Net Storage and Retrieval Statements

a. QPUT

The format of the QPUT statement is:

(QPUT p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$})

The steps in the evaluation of a QPUT statement are as follows:

(1) The pattern p-exp is instantiated, and an expression exp that matches the instantiation is retrieved from the net.

(2) A context CTX in which to evaluate the statement is determined as described above.

(3) The indicator-property pairs are instantiated, and all properties $prop_i$ are assigned to the expression under the corresponding indicators ind_i with respect to CTX.

(4) If the NAME option is present, then the statement is named name. Otherwise the statement is named QPUT.

(5) If there is an APPLY option, team is instantiated and all of its functions are applied successively to

exp. The default for context references in the team functions will be CTX.
If any team function fails, its side effects are undone.
The functions in the team may cause the QPUT statement itself to fail, as described in Section II-G, above.

(6) The statement returns exp as its value. For example, in the sample program in Annex A. the QPUT statement in HITCH,
(QPUT(PERSON $HUMAN)MARRIEDTO $Y APPLY $COMPUTERELATIONS)
operates as follows when the argument to HITCH is (HAPPY ADAM)

- The pattern (PERSON $HUMAN) is instantiated to (PERSON ADAM), and (PERSON ADAM) is retrieved from the net.
- The statement will be evaluated in the top context, because no context was specified and the current default context is the top context.
- $Y is instantiated to ADAMs spouse, say, EVE, and placed as the value of the indicator MARRIEDTO on the property list of (PERSON ADAM).
- The statement is named "QPUT."
- The team $COMPUTERELATIONS is instantiated to the tuple (MAKESPOUSE).
  The function MAKESPOUSE is called with the argument (PERSON ADAM); if the (FAIL CALLER) statement in MAKESPOUSE is executed, the QPUT statement itself will fail and the property list of (PERSON ADAM) will be restored.
  If a (FAIL) statement instead of (FAIL CALLER) had been executed, then the side effects of MAKESPOUSE would have been undone but execution of the QPUT statement would have continued.
- (PERSON ADAM) is returned as the value of the QPUT statement.

b. QGET

The format of the QGET statement is:

(QGET p-exp {APPLY team} {WRT ctx} {NAME name}

$$\left[\begin{array}{l} \text{ind} \\ \text{ind}_1\ \text{prop}_1\ \{\text{ind}_2\ \text{prop}_2\ \ldots\ \text{ind}_n\ \text{prop}_n\} \end{array}\right]\ )$$

The evaluation of a QGET statement is similar to that of QPUT, except that it retrieves values for each indicator ind_i. If no property can be found for an indicator, NOSUCHPROPERTY is used as its value. The values are matched against the corresponding patterns prop_i. If a match for p-exp cannot be found, or if the match to some prop_i does not succeed, the QGET statement fails.

If only one indicator (and no properties) was specified, QGET returns the corresponding property value. Otherwise, QGET returns the expression exp.

For example, in the sample program in Appendix A, when the QGET statement in CHECKAGE,

(QGET (PERSON (@ SPOUSE)) SEX ←SEX AGE ←AGE)

is called with EVE as the value of SPOUSE, then $SEX will be bound to her sex, $AGE will be bound to her age, and (PERSON EVE) will be returned as the value of the QGET statement.

c. ASSERT

The format of the ASSERT statement is:

(ASSERT p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 prop_1 ... ind_n prop_n})

ASSERT performs (QPUT p-exp {APPLY team} {WRT ctx} {NAME name}
MODELVALUE T {ind_1 prop_1 ... ind_n prop_n}).

Responsibility for consistency checks rests with the functions in the APPLY team.

d. DENY

The format of the DENY statement is:

(DENY p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$})

DENY performs (QPUT p-exp {APPLY team} {WRT ctx} {NAME name}
MODELVALUE NIL {ind_1 $prop_1$ ... ind_n $prop_n$})

Responsibility for consistency checks rests with the functions of the APPLY team.

e. IS

The format of the IS statement is:

(IS p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$})

IS performs (QGET p-exp {APPLY team} {WRT ctx} {NAME name}
MODELVALUE T {ind_1 $prop_1$ ... ind_n $prop_n$}).

f. ISNT

The format of the ISNT statement is:

(ISNT p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$})

ISNT performs (QGET p-exp {APPLY team} {WRT ctx} {NAME name}
MODELVALUE NIL {ind_1 $prop_1$ ... ind_n $prop_n$})

g. BIS

The format of the BIS statement is:

(BIS p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$} THEN e_1 {e_2 ... e_m})

BIS performs the same retrieval as IS, but when an expression has been found and the team members applied, a backtrack point is established and the expressions $e_1 \ldots e_m$ are evaluated in turn. If a failure occurs, BIS will continue to retrieve different expressions from the net until either none of the e_i fails, in which case BIS returns the retrieved expression, or until all possible retrievals from the net have been attempted, in which case BIS fails.

BIS is a temporary expedient, which may be removed from the language when INTERLISP permits control structure backtracking. At that time the IS statement will be able to establish a backtrack point, and failures below it will cause another attempt at retrieval to take place.

For example, in the sample program of Appendix A, when HITCH is called with the argument (HAPPY ADAM), the BIS statement will retrieve an expression from the net of the form (PERSON ←Y) that has the property FEMALE under the indicator SEX. The statements after THEN will be evaluated. If any statement fails, another expression will be retrieved from the net, and the cycle will be repeated. If no statement fails BIS will return (HAPPY ADAM). If no expression had been found in the net for which none of the statements failed, then the BIS statement would have failed.

h. <u>INSTANCES</u>

The format of the INSTANCES statement is:

(INSTANCES <u>p-exp</u> {APPLY <u>team</u>} {WRT <u>ctx</u>} {NAME <u>name</u>} {ind_1 $prop_1$ ... ind_n $prop_n$})

INSTANCES instantiates the pattern <u>p-exp</u>, determines a context CTX, and computes a <u>name</u> as was described for the QPUT statement.

INSTANCES then retrieves all the expressions from the net that match the instantiation of <u>p-exp</u>, that have the value

indicator MODELVALUE (unless some ind_i is MODELVALUE) and that have properties on the indicators ind_i which match the property patterns $prop_i$, with respect to the context CTX.

For each such expression found, all the members of the APPLY team are applied to it successively.

A CLASS of all the retrieved expressions is returned as the value of the statement.

i. GOAL

The format of the GOAL statement is:

(GOAL p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$}).

GOAL first performs (IS p-exp {WRT ctx} {NAME name}
{ind_1 $prop_1$ ... ind_n $prop_n$}).

If an expression was found, it is returned as the value of the statement. If not, the functions of the team are applied successively to the instantiated p-exp until some team member does not fail. The value returned by that team member is then returned as the value of the statement. If all the functions of the team fail, the GOAL statement fails.

j. CASES

The format of the CASES statement is:

(CASES p-exp {APPLY team} {WRT ctx} {NAME name}).

CASES is equivalent to GOAL, except that the IS statement is not performed.

k. DELETE

The format of the DELETE statement is:

(DELETE p-exp {APPLY team} {WRT ctx} {NAME name}
{ind_1 prop_1 ... ind_n prop_n})

DELETE performs the equivalent of

(INSTANCES p-exp {APPLY team} {WRT ctx} {NAME name}
MODELVALUE (POR T NIL) {ind_1 prop_1 ... ind_n prop_n})

For each expression of the CLASS returned by INSTANCES, the indicator MODELVALUE is removed. DELETE returns that CLASS.

3. Other Statements That Can Set a Default Context

a. DO

The format of the DO statement is:

(DO tpl WRT ctx)

DO instantiates tpl in the current context. The pattern tpl must instantiate to a pattern of STYPE TUPLE. The WRT clause is evaluated, and this determines a new default context. The instantiation of tpl is evaluated in this context, and the result of the evaluation is returned as the value of DO.

b. MATCH

The format of the MATCH statement is:

(MATCH exp val {WRT ctx})

MATCH evaluates exp and val. It then attempts to match the value of val to the value of exp. Any new variable bindings that are created by the matching process are made with respect to ctx, if present, or to the current context. The value of MATCH is the value of val.

c. MATCHQ

MATCHQ is identical to MATCH except that the argument exp is instantiated rather than evaluated.

This statement corresponds to the SETQ statement in QA4.

d. MATCHQQ

MATCHQQ is identical to MATCH except that both exp and val are instantiated rather than evaluated.

This statement corresponds to the SETQQ statement in QA4.

III HOW TO USE QLISP

A. Loading the System

The QLISP system may be loaded into LISP by typing:

SYSIN(⟨SACERDOTI⟩QLISP.SYS) .

All subsequent type-ins will be processed by the full QLISP system.

B. Creating and Using Symbolic Files

QLISP uses the INTERLISP file package to manipulate symbolic files. PRETTYDEF, MAKEFILE, LOAD, and LOADFNS all know about QLISP constructs and handle them properly. Variables whose values are to be extracted from the net on output and stored into the net on input should be prefixed by a $. Input to and output from the net is done with respect to the current dynamic context at the time of the I/O operation. Input of net variables is done with respect to the context ETERNAL, and thus the values of LOADed variables are not backtrackable (although the effect of a LOAD is UNDOable if it is initiated at the teletype).

C. Defining Functions

QLISP functions are defined by using the functions PUTD, DEFINE, and DEFINEQ.

D. Editing Functions and Variables

All QLISP functions may be edited by using EDITF.

QLISP variables may be edited by using EDITV. Net variables should be prefixed by a $. Retrieval from and storage into the net is done

with respect to the current dynamic context at the time EDITV is called. If the value of a net variable that is not at the top context is being edited, EDITV will print a warning message.

E. Tracing QLISP Functions

Since the current implementation of LISP does not permit us to backtrack properly through BREAKs, we have implemented a trace facility that ADVISEs rather than BREAKs functions.

Functions may be traced by executing the function QTRACE. It is an NLAMBDA no-spread function (just like TRACE), and thus can accept a single function name or a sequence of function names.

The tracing of functions may be turned off by invoking the function UNQTRACE, which is also an NLAMBDA no-spread. Calling UNQTRACE with no arguments causes all traced functions to be untraced.

When new functions are defined by loading a symbolic file or by explicit calls to PUTD, DEFINE or DEFINEQ, all QLAMBDA functions will be automatically QTRACEd if the global variable QTRACEALL is set to T. (It is set to T when QLISP is first loaded.)

QTRACE output may be directed to a file other than the teletype by executing the command TRFILE (file name). This will also set LINELENGTH to 120 to conserve line printer paper. The command UNTRFILE () will redirect QTRACE output back to the teletype, reset LINELENGTH, and close the previous output file.

F. Restrictions and Caveats

All function, variable, and property names beginning with "QA4:" are reserved for the QLISP system.

LISP variables should not begin with $, ?, or ←.

With one exception, all CLISP constructs are valid in QLISP. The exception is the use of the quote mark ('), which is used in QLISP to denote the quasi-quote rather than the LISP QUOTE.

ACKNOWLEDGMENTS

In creating QLISP, we have assembled in one package the good ideas of many individuals. We are particularly indebted to Richard Waldinger, Richard Fikes, Jeff Rulifson, Warren Teitelman, and Mark Stickel. The development of the QLISP language was supported by the National Aeronautics and Space Administration under Contract NASW-2086. This work was made possible by the environment and facilities of the SRI Artificial Intelligence Center, which has been largely supported by the Advanced Research Projects Agency through Contract DAHC04-72-C-0008.

REFERENCES

1. J. Rulifson, Derksen, J. A., and Waldinger, R. W., "QA4: A Procedural Calculus for Intuitive Reasoning," AIC Technical Note 73, Stanford Research Institute, Menlo Park, California (November 1972).

2. W. Teitelman, et al., BBN-LISP TENEX Reference Manual, Bolt Beranek and Newman, Cambridge, Massachusetts (July 1971).

3. W. Teitelman, "CLISP--Conversational LISP," Third International Joint Conference on Artificial Intelligence, Advance Papers of the Conference, pp. 686-690, Stanford Research Institute, Menlo Park, California, August 1973).

4. Teitelman, W., et al., op. cit., pp. 686-690.

Appendix A

A SAMPLE QLISP PROGRAM

Appendix A

A SAMPLE QLISP PROGRAM

With the goal of creating a better world through computer science, we present below a small system for making people happy. It was written not with elegance or efficiency in mind, but to give examples of the new QLISP features and their interactions with INTERLISP.

A. Program Testing

This symbolic file was created by MAKEFILE in the ordinary fashion.

```
;   <REBOH>GENESIS.;3    MON 13-AUG-73 2:25PM


(FILECREATED "19-JUL-73 16:44:07" GENESIS)


(DEFINEQ

(SETUP
  [LAMBDA NIL                                          (* INITIALIZATION
                                                          ROUTINE.)
    (ASSERT (PERSON MARY)
            SEX FEMALE AGE 30 HOBBIES (CLASS TENNIS NEEDLEPOINT DANCING)
            )
    (ASSERT (PERSON ALICE)
            SEX FEMALE AGE 72 HOBBIES (CLASS SCUBA-DIVING BIRD-WATCHING)
            )
    (ASSERT (PERSON EVE)
            SEX FEMALE AGE 29 HOBBIES (CLASS SNAKE-CHARMING GARDENING
                                             VOLLEYBALL))
    (ASSERT (PERSON ADAM)
            SEX MALE AGE 30 NETWORTH 500000 HOBBIES
            (CLASS HUNTING FISHING GARDENING))
    (ASSERT (PERSON SARA)
            SEX FEMALE AGE 42 NETWORTH 2000000])

(MAKEHAPPY
  [LAMBDA (L)                                          (* 'L IS A LIST OF
                                                          PERSONS.)
                                                       (* TRY TO MAKE EACH
                                                          PERSON HAPPY.)
    [MAPC L (FUNCTION (LAMBDA (X)
              (PRINT (ATTEMPT (GOAL (HAPPY (@ X))
                                    APPLY
                                    (TUPLE HITCH RICH]
    (QUOTE FINISHED])

(HITCH
  [QLAMBDA (HAPPY ←HUMAN)                              (* CYCLE THROUGH ALL
                                                          MEMBERS OF THE OPPOSITE
                                                          SEX.)
                                                       (* HYPOTHESIZE A
                                                          MARRIAGE AND SEE IF IT
                                                          WORKS OUT.)
                                                       (* IF IT DOES, THEN THE
                                                          HUMAN IS HAPPY.)
           (BIS (PERSON ←Y)
                SEX
                [@ (PARTNERSEX ('(PERSON $HUMAN]
               THEN (ASSERT (MARRIED $HUMAN $Y)
                            APPLY $MARRIAGEDEMONS)
                    (QPUT (PERSON $HUMAN)
                          MARRIEDTO $Y WRT GLOBAL APPLY
                          $COMPUTERELATIONS)
                    ('(HAPPY $HUMAN])
```

```
;  <REBOH>GENESIS.;3   MON 13-AUG-73 2:25PM

(CHECKAGE
  (QLAMBDA (MARRIED ←COUPLE)                          (* MAKE SURE THE WIFE IS
                                                      NOT TOO MUCH OLDER THAN
                                                      THE HUSBAND.)
          (QPROG (←SEX
                  ←AGE
                  MALEAGE FEMALEAGE)
                 [MAPC (CDR $COUPLE)
                       (FUNCTION (LAMBDA (SPOUSE)
                            (QGET (PERSON (@ SPOUSE))
                                  SEX ←SEX
                                  AGE ←AGE)
                          (IF (EQ $SEX (QUOTE MALE))
                              THEN (SETQ MALEAGE $AGE)
                            ELSE (SETQ FEMALEAGE $AGE]
                 (IF (GREATERP FEMALEAGE (PLUS MALEAGE 5))
                     THEN (FAIL CALLER))
                 (QRETURN OK))))

(CHECKHOBBY
  (QLAMBDA (MARRIED ←X
                    ←Y)                               (* FIND AT LEAST ONE
                                                      HOBBY IN COMMON.
                                                      OTHERWISE FAIL.)
          (ATTEMPT (MATCHQQ (TUPLE (CLASS ←H
                                          ←←OTHERS)
                                   (CLASS ←H
                                          ←←OTHEROTHERS))
                            (TUPLE (QGET (PERSON $X)
                                         HOBBIES)
                                   (QGET (PERSON $Y)
                                         HOBBIES)))
            ELSE (FAIL CALLER))))

(PARTNERSEX
  (QLAMBDA ←X                                         (* FIND THE OPPOSITE SEX
                                                      OF THE PERSON IN
                                                      QUESTION.)
          (SELECTQ (QGET $X SEX)
                   (MALE (QUOTE FEMALE))
                   (FEMALE (QUOTE MALE))
                   (ERROR "UNKNOWN SEX ")))))
```

```
;  <REBOH>GENESIS.;3    MON 13-AUG-73 2:25PM

1  (RICH
     (QLAMBDA (HAPPY ←HUMAN)                        (* TRY TO ACHIEVE A NET
                                                    WORTH GREATER THAN ONE
                                                    MILLION.)
                                                    (* IF ACHIEVABLE, THEN
                                                    THE HUMAN IS HAPPY.)
                                                    (* THIS ROUTINE NOW ONLY
                                                    MAKES A SIMPLE CHECK
                                                    AGAINST THE DATA BASE.)
              (IF (GREATERP (GGET (PERSON $HUMAN)
                                  NETWORTH)
                            1000000)
                  THEN ('(HAPPY $HUMAN))
                ELSE (FAIL))))

   (MAKESPOUSE
     [QLAMBDA (PERSON ←PERSON)                      (* TEAM MEMBER OF
                                                    $COMPUTERELATIONS.)
                                                    (* ENSURES THAT THE
                                                    SPOUSE IS NOT ALREADY
                                                    MARRIED.)
                                                    (* ASSERTS THAT THE
                                                    SPOUSE IS MARRIED.)
              (QPROG ((←SPOUSE
                        (QGET (PERSON $PERSON)
                              MARRIEDTO)))
                     (IF (NOT (EQ (QGET (PERSON $SPOUSE)
                                        MARRIEDTO)
                                  (QUOTE NOSUCHPROPERTY)))
                         THEN (FAIL CALLER)
                       ELSE (QPUT (PERSON $SPOUSE)
                                  MARRIEDTO $PERSON])
   )
     (LISPXPRINT (QUOTE GENESISFNS)
                 T)
     (RPAQQ GENESISFNS (SETUP MAKEHAPPY HITCH CHECKAGE CHECKHOBBY
                              PARTNERSEX RICH MAKESPOUSE))
     (LISPXPRINT (QUOTE GENESISVARS)
                 T)
     [RPAQQ GENESISVARS ($MARRIAGEDEMONS $COMPUTERELATIONS
                                     (P (QSETUP GENESISVARS))
                                     (P (DEFTYPE (QUOTE MARRIED)
                                                 (QUOTE CLASS]
     (RPAQQ $MARRIAGEDEMONS (CHECKAGE CHECKHOBBY))
     (RPAQQ $COMPUTERELATIONS (MAKESPOUSE))
     (QSETUP GENESISVARS)
     (DEFTYPE (QUOTE MARRIED)
              (QUOTE CLASS))
   STOP
```

B. Sample GENESIS Run

Here is a session at the teletype using the functions of the GENESIS file in the QLISP system:

```
@LISP

INTERLISP-10  07-07-73 ...


HI, RENE.
←SYSIN(<SACERDOTI>QLISP.SYS]             Load in QLISP
(<SACERDOTI>QLISP.SYS;20)
QHELLO
←LOAD(GENESIS]                           Load file with user programs
FILE CREATED 19-JUL-73 16:44:07          Since the variable QTRACEALL is set to T,
GENESISFNS                                  all QLAMBDA functions will be QTRACEd
GENESISVARS
GENESIS.;3
←$MARRIAGEDEMONS                         Net variables are treated just like
(CHECKAGE CHECKHOBBY)                         LISP variables
←PP(HITCH]                               QLAMBDA expressions are treated just like
                                              LAMBDA expressions
(HITCH
  [QLAMBDA (HAPPY ←HUMAN)  **COMMENT**    **COMMENT**    **COMMENT**
          (BIS (PERSON ←Y)
               SEX
               [@ (PARTNERSEX ('(PERSON $HUMAN]
              THEN (ASSERT (MARRIED $HUMAN $Y)
                          APPLY $MARRIAGEDEMONS)
                   (QPUT (PERSON $HUMAN)
                         MARRIEDTO $Y WRT GLOBAL APPLY
                         $COMPUTERELATIONS)
                   ('(HAPPY $HUMAN])
(HITCH)
←SETUP]                                  Evaluate the user's initialization function
T
```

```
←MAKEHAPPY((ADAM SARA))                      Try to make Adam and Sara happy
HITCH:
QA4:ARG= (HAPPY ADAM)
  PARTNERSEX:                           ———Example of function trace
  QA4:ARG= (PERSON ADAM)
  (PARTNERSEX) = FEMALE                 ———Function returns 'FEMALE
  CHECKAGE:
  QA4:ARG= (MARRIED ADAM MARY)
  (CHECKAGE) = OK
  CHECKHOBBY:                           ———Function is called, but does
  QA4:ARG= (MARRIED ADAM MARY)             not return; it caused a failure
  CHECKAGE:
  QA4:ARG= (MARRIED ADAM ALICE)
  CHECKAGE:
  QA4:ARG= (MARRIED ADAM EVE)
  (CHECKAGE) = OK
  CHECKHOBBY:
  QA4:ARG= (MARRIED ADAM EVE)
  (CHECKHOBBY) = ((CLASS SNAKE-CHARMING GARDENING VOLLEYBALL) (CLASS
GARDENING HUNTING FISHING))
  MAKESPOUSE:
  QA4:ARG= (PERSON ADAM)
  (MAKESPOUSE) = ADAM
(HITCH) = (HAPPY ADAM)
(HAPPY ADAM)
HITCH:
QA4:ARG= (HAPPY SARA)
  PARTNERSEX:
  QA4:ARG= (PERSON SARA)
  (PARTNERSEX) = MALE
  CHECKAGE:
  QA4:ARG= (MARRIED ADAM SARA)
RICH:
QA4:ARG= (HAPPY SARA)

GC: 8                                        Garbage collection of list storage
5980, 10068 FREE WORDS
(RICH) = (HAPPY SARA)
(HAPPY SARA)
FINISHED
←
```

Appendix B

THE SYNTAX OF QLISP STATEMENTS

Appendix B

THE SYNTAX OF QLISP STATEMENTS

For the syntax of INTERLISP statements see reference 3.

The representation of the syntax of the other QLISP statements is presented below. The following notation is used:

- Braces ({ }) indicate that the enclosed elements may be omitted.
- Square brackets ([ ]) indicate the choice of one of the enclosed elements.
- A subscript indicates an element of a sequence.
- e indicates an expression to be evaluated.
- p-exp indicates an expression to be instantiated.
- tpl indicates an expression of STYPE TUPLE.
- var indicates a net variable.
- ctx indicates an expression that instantiates to a context.
- name indicates an expression that instantiates to a statement name.
- ind indicates an expression that instantiates to a property list indicator.
- prop indicates an expression that instantiates to a property.
- tuple, vector, bag, and class indicate expressions that evaluate to a TUPLE, VECTOR, BAG, or CLASS respectively. Note that a LISP list is treated as a tuple.

$$\texttt{(ASSERT p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \{\texttt{WRT ctx}\}\ \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\}\texttt{)}$$

$$\texttt{(ATTEMPT } e_1\{e_2\ \ldots\ e_n\}\ \{\texttt{THEN } e'_1\ \ldots\ e'_m\}\ \{\texttt{ELSE } e''_1\ \ldots\ e''_k\}\texttt{)}$$

$$\texttt{(BIS p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \{\texttt{WRT ctx}\}\ \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\}\ \{\texttt{THEN } e_1\ \ldots\ e_n\}\texttt{)}$$

$$\texttt{(CASES p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \{\texttt{WRT ctx}\}\ \{\texttt{name name}\}\texttt{)}$$

$$\texttt{(CONTEXT} \begin{bmatrix} \texttt{PUSH} \\ \texttt{POP} \\ \texttt{CURRENT} \end{bmatrix} \texttt{ctx)}$$

$$\texttt{(DELETE p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \{\texttt{WRT ctx}\}\ \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\}\texttt{)}$$

$$\texttt{(DENY p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \{\texttt{WRT ctx}\}\ \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\}\texttt{)}$$

(DO tpl WRT ctx)

$$\texttt{(FAIL} \left\{ \begin{bmatrix} \texttt{CALLER} \\ \texttt{name} \end{bmatrix} \right\} \texttt{)}$$

$$(\texttt{ASSERT p-exp} \left\{ \texttt{APPLY} \begin{bmatrix}\texttt{tuple}\\ \texttt{vector}\\ \texttt{bag}\\ \texttt{class}\end{bmatrix} \right\} \{\texttt{WRT ctx}\} \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\})$$

$$(\texttt{ATTEMPT}\ e_1\{e_2\ \ldots\ e_n\}\ \{\texttt{THEN}\ e'_1\ \ldots\ e'_m\}\ \{\texttt{ELSE}\ e''_1\ \ldots\ e''_k\})$$

$$(\texttt{BIS p-exp} \left\{ \texttt{APPLY} \begin{bmatrix}\texttt{tuple}\\ \texttt{vector}\\ \texttt{bag}\\ \texttt{class}\end{bmatrix} \right\} \{\texttt{WRT ctx}\} \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\}\ \{\texttt{THEN}\ e_1\ \ldots\ e_n\})$$

$$(\texttt{CASES p-exp} \left\{ \texttt{APPLY} \begin{bmatrix}\texttt{tuple}\\ \texttt{vector}\\ \texttt{bag}\\ \texttt{class}\end{bmatrix} \right\} \{\texttt{WRT ctx}\} \{\texttt{name name}\})$$

$$(\texttt{CONTEXT} \begin{bmatrix}\texttt{PUSH}\\ \texttt{POP}\\ \texttt{CURRENT}\end{bmatrix} \texttt{ctx})$$

$$(\texttt{DELETE p-exp} \left\{ \texttt{APPLY} \begin{bmatrix}\texttt{tuple}\\ \texttt{vector}\\ \texttt{bag}\\ \texttt{class}\end{bmatrix} \right\} \{\texttt{WRT ctx}\} \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\})$$

$$(\texttt{DENY p-exp} \left\{ \texttt{APPLY} \begin{bmatrix}\texttt{tuple}\\ \texttt{vector}\\ \texttt{bag}\\ \texttt{class}\end{bmatrix} \right\} \{\texttt{WRT ctx}\} \{\texttt{NAME name}\}$$

$$\{\texttt{ind}_1\ \texttt{prop}_1\ \ldots\ \texttt{ind}_n\ \texttt{prop}_n\})$$

$$(\texttt{DO tpl WRT ctx})$$

$$(\texttt{FAIL} \left\{ \begin{bmatrix}\texttt{CALLER}\\ \texttt{name}\end{bmatrix} \right\})$$

$$\texttt{(GOAL p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \texttt{\{WRT ctx\} \{NAME name\}}$$

$$\texttt{\{ind}_1 \texttt{ prop}_1 \ \ldots \ \texttt{ind}_n \texttt{ prop}_n\texttt{\})}$$

$$\texttt{(INSTANCES p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \texttt{\{WRT ctx\} \{NAME name\}}$$

$$\texttt{\{ind}_1 \texttt{ prop}_1 \ \ldots \ \texttt{ind}_n \texttt{ prop}_n\texttt{\})}$$

$$\texttt{(IS p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \texttt{\{WRT ctx\} \{NAME name\}}$$

$$\texttt{\{ind}_1 \texttt{ prop}_1 \ \ldots \ \texttt{ind}_n \texttt{ prop}_n\texttt{\})}$$

$$\texttt{(ISNT p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \texttt{\{WRT ctx\} \{NAME name\}}$$

$$\texttt{\{ind}_1 \texttt{ prop}_1 \ \ldots \ \texttt{ind}_n \texttt{ prop}_n\texttt{\})}$$

$$\texttt{(MATCH e}_1 \texttt{ e}_2 \texttt{ \{WRT ctx\})}$$

$$\texttt{(MATCHQ p-exp e \{WRT ctx\})}$$

$$\texttt{(MATCHQQ p-exp}_1 \texttt{ p-exp}_2 \texttt{ \{WRT ctx\})}$$

$$\texttt{(QGET p-exp} \left\{ \texttt{APPLY} \begin{bmatrix} \texttt{tuple} \\ \texttt{vector} \\ \texttt{bag} \\ \texttt{class} \end{bmatrix} \right\} \texttt{\{WRT ctx\} \{NAME name\}}$$

$$\begin{bmatrix} \texttt{ind} \\ \texttt{ind}_1 \texttt{ prop}_1 \texttt{ \{ind}_2 \texttt{ prop}_2 \ \ldots \ \texttt{ind}_n \texttt{ prop}_n\texttt{\}} \end{bmatrix}\texttt{)}$$

$$\texttt{(QPROG args e}_1 \texttt{ e}_2 \ \ldots \ \texttt{e}_n\texttt{)}$$

$$\text{(QPUT p-exp } \left\{ \text{APPLY} \begin{bmatrix} \text{tuple} \\ \text{vector} \\ \text{bag} \\ \text{class} \end{bmatrix} \right\} \{\text{WRT ctx}\} \; \{\text{NAME name}\}$$

$$\text{ind}_1 \text{ prop}_1 \; \{\text{ind}_2 \text{ prop}_2 \; \ldots \; \text{ind}_n \text{ prop}_n\})$$

(QRETURN {p-exp})

(STYPE e)

(VAL var {WRT ctx})